





Prática de Programação C# com criação de classes

Já vimos bastante coisa sobre C#, mas ainda faltam alguns pequenos e importantes detalhes. Nessa aula abordaremos o tema Prática de Programação C# II, mas para isso iremos percorrer a seguinte trilha pedagógica:

- **Classes e Objetos**
- **Modificadores e Construtores**
- **This e Base**
- **Encapsulamento e Polimorfismo**

- **Classes e Objetos**

Uma classe em C# é definido com o nome "class" e é na verdade um tipo de referência, ou seja, quando você declara uma variável de classe, ela terá o valor nulo até que você crie "explicitamente uma instância da classe usando o novo operador ou atribua a ela um objeto de um tipo compatível que pode ter sido criado em outro lugar". (MICROSOFT, 2020).

Por exemplo, imaginamos que precisaríamos criar personagens para o game que seriam monstros. Poderíamos colocar as características básicas do monstro, porém cada personagem poderia ter características específicas e/ou modificações (nome, força, cor, espécie etc.), algumas coisas mudariam, mas continuariam monstros. Para isso eu posso usar uma classe (Figura 1- a).



Figura 1 – Classe e Objetos de Monstros | Fonte: Autor, 2022.

Agora que criamos a classe é a ora de criarmos os objetos, afinal, lembra-se que o C# é uma programação orientada ao objeto. Um objeto nada mais é que uma instância da Classe (Figura 1- b).



> Embora eles sejam usados algumas vezes de maneira intercambiável, uma classe e um objeto são coisas diferentes. Uma classe define um tipo de objeto, mas não é um objeto em si. Um objeto é uma entidade concreta com base em uma classe e, às vezes, é conhecido como uma instância de uma classe. (MICROSOFT, 2022)

Os objetos devem ser declarados, mas para serem criados é necessário acrescentar a instância "new" seguida pelo nome da classe na qual o objeto se baseará, no nosso exemplo (Figura 1 – c) "saciFase1 = new monstro ();" e "saciFase2 = new monstro ();".

Observe que na classe eu tenho um tipo de "receita de bolo" de como meu "Monstro" deve ser, depois posso ir acrescentando elementos e modificadores no modelo básico do "bolo".

- **Modificadores e Construtores**

Para que a classe seja útil é necessário que tenhamos conhecimentos sobre Modificadores de acesso. Agora vamos falar sobre os principais modificadores que são (MICROSOFT, 2022):

- **public:** O acesso não é restrito;
- **protected:** O acesso é limitado à classe que a contém ou aos tipos derivados da classe que a contém;
- **internal:** O acesso é limitado ao assembly atual;
- **protected internal:** O acesso é limitado ao assembly ou tipos atuais derivados da classe que a contém;
- **private:** O acesso é limitado ao tipo recipiente;
- **private protected:** O acesso é limitado à classe que a contém ou aos tipos derivados da classe que a contém dentro do assembly atual.



Os modificadores são importantes para que possamos classificar os objetos em outra instância dentro do código, ou seja, caso eu use o modificador "private" ou "protected" eu não poderia colocar o objeto em outra classe além da qual ele está vinculado ou a uma irmã (Figura 2).

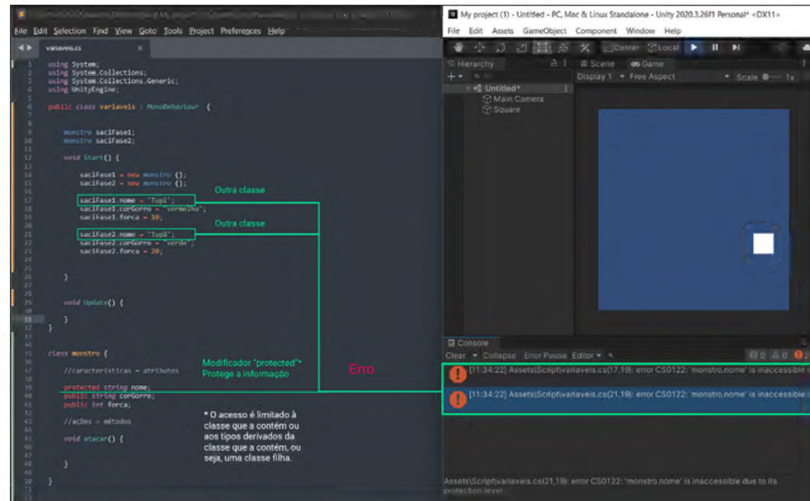


Figura 2 – Modificador protected | Fonte: Autor, 2022.

Para aparecer em outra classe é necessário que o modificador esteja em public. Como é o caso do elemento "public string corGorro;" no exemplo da Figura 2.

Agora vamos falar sobre o método construtor. Os construtores são tipos especiais de métodos usados para criar e inicializar objetos. É através deste tipo especial de método, chamado de construtor, que você cria instâncias de uma classe, ou seja, você pode colocar um valor padrão inicial de um determinado atributo, por exemplo.

Você pode trabalhar esse método de duas formas:



1. Colocando valores na hora de criar o seu objeto	2. Colocando os valores dentro da classe
{ metodos obj; void Start () { obj = new metodos (20, "olá pessoal!"); print(obj.valor1); print(obj.valor2); } void Update () {} } class metodos { public int valor1; public string valor2; public metodos (int valor, string val) { this.valor1 = valor; this.valor2 = val; } }	{ metodos obj; void Start () { obj = new metodos (); print(obj.valor1); print(obj.valor2); } void Update () {} } class metodos { public int valor1; public string valor2; public metodos () { valor1 = 10; valor2 = "Oi"; } }

Tabela 1 – Tipos de Construtores | Fonte: NASCIMENTO, 2022.

Uma outra forma de se ter um método é o estático. "Uma classe ou struct também pode ter um construtor estático, que inicializa membros estáticos do tipo. Construtores estáticos não têm parâmetros" (MICROSOFT, 2022), ou seja, não possui um objeto:

```
{
    void Start ()
    {
        int retorno = metodos.soma (2,7);
        print (retorno);
    }


    void Update () {}
}
```

```
class metodos

{

  public static int soma (int valor1, int valor2)

  {

    int result = valor1 + valor2;

    return result;

  }

}
```



- **This e Base**

Agora iremos falar sobre "this" e "base". Você deve estar se perguntando, para que serve cada um desses elementos?

Vamos lá:

- **This:** serve para referenciar o próprio objeto (figura 3);
- **Base:** serve para acessar os membros da classe base dento de uma classe derivada (figura 4).

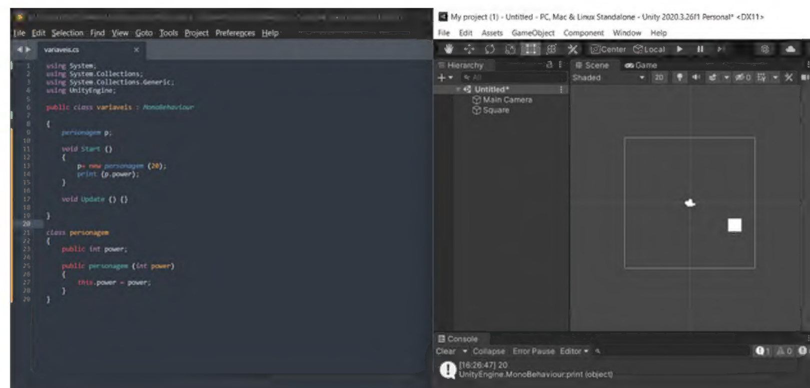


Figura 3 – This | Fonte: Autor, 2022.

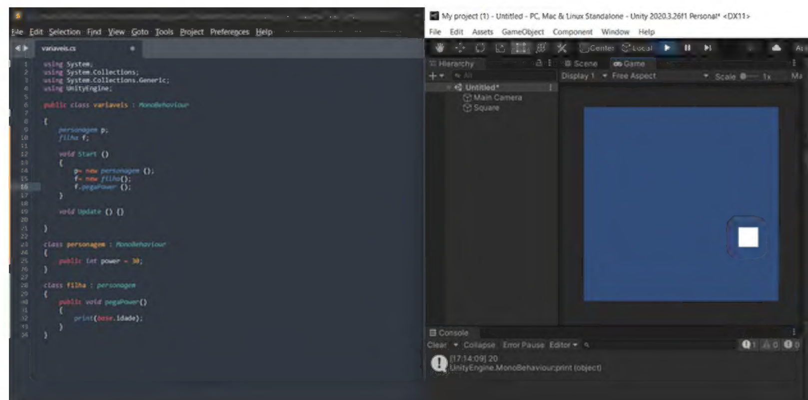


Figura 4 – Base | Fonte: Autor, 2022.

4. **Encapsulamento, Herança e Polimorfismo**

Já falamos que C# é uma linguagem de programação orientada a objetos, várias vezes em nossas aulas, porém, o que pode ser novo é que existem 4 princípios básicos da programação orientada a objetos (MICROSOFT, 2022):

- **Abstração:** os atributos relevantes e as interações de entidades como classes para definir uma representação abstrata de um sistema.

- **Encapsulamento:** ocultar o estado interno e a funcionalidade de um objeto e permitir o acesso apenas por meio de um conjunto público de funções.

- **Herança:** capacidade de criar abstrações com base em abstrações existentes.

- **Polimorfismo:** capacidade de implementar propriedades ou métodos herdados de diferentes maneiras em várias abstrações.

4.1 Encapsulamento

A abstração já foi abordada significativamente nos elementos apresentados até aqui. Já o Encapsulamento (figura 5), que é o processo de usar modificadores de acesso para ocultar ou esconder membros de uma classe de acesso externo, não foi tão explorado. Portanto, o encapsulamento fornece um meio de preservar a integridade do estado dos dados. Para que isso ocorra, em vez de definir campos públicos, devemos definir campos de dados privados.

Uma classe bem encapsulada deve ocultar seus dados e detalhes de implementação do mundo exterior. Isso é chamado de programação de caixa preta.

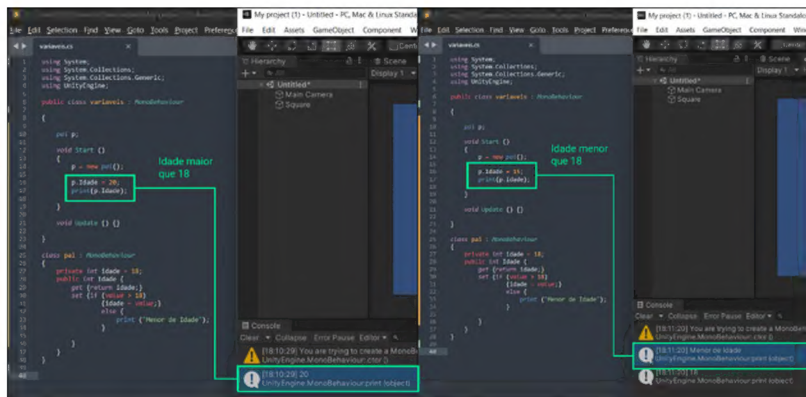


Figura 5 – Encapsulamento | Fonte: Autor, 2022.

4.2 Herança

É por meio da herança que uma classe é capaz de herdar todas as suas propriedades, atributos e métodos de outra classe. A classe que possibilita o compartilhamento de suas propriedades é chamada de classe pai, já a que herda é a classe filha ou derivada. Para declarar a herança é bem simples, é só usar o ":" (figura 6).

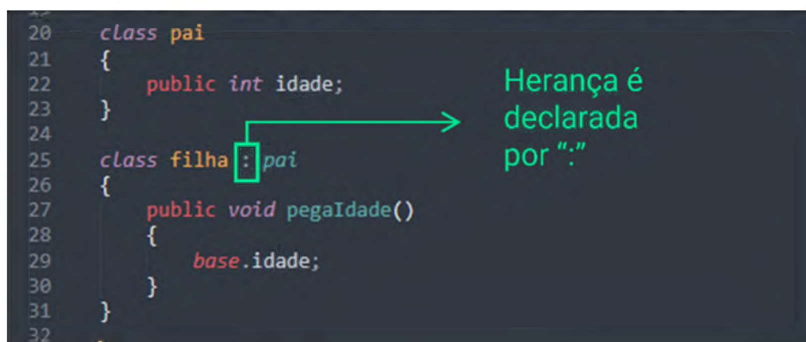


Figura 6 – Herança | Fonte: Autor, 2022.

4.3 Polimorfismo

Finalizando o conteúdo sobre C# vamos falar sobre Polimorfismo. Já vimos que Polimorfismo é a capacidade das classes de fornecer diferentes implementações de métodos que são chamados pelo mesmo nome.

Segundo o Guia do C# da Microsoft (2022), o polimorfismo é uma palavra grega que significa "de muitas formas" e tem dois aspectos distintos, vejamos o exemplo dado pela empresa de tecnologia:

> Os métodos virtuais permitem que você trabalhe com grupos de objetos relacionados de maneira uniforme. Por exemplo, suponha que você tem um aplicativo de desenho que permite que um usuário crie vários tipos de formas sobre uma superfície de desenho. Você não sabe no tempo de compilação quais tipos específicos de formas o usuário criará. No entanto, o aplicativo precisa manter controle de todos os diferentes tipos de formas que são criados e atualizá-los em resposta às ações do mouse do usuário. Você pode



usar o polimorfismo para resolver esse problema em duas etapas básicas: 1) Crie uma hierarquia de classes em que cada classe de forma específica derive de uma classe base comum; 2) Use um método virtual para invocar o método adequado em qualquer classe derivada por meio de uma única chamada para o método da classe base.

Veja a seguir o código proposto pela Microsoft para resolução do problema proposto anteriormente:

E:\E-mail\2022\Janeiro\Descomplica\My project (1)\Assets\Script\variaveis.cs • - Sublime T

File Edit Selection Find View Goto Tools Project Preferences Help

variaveis.cs

```
public class Shape
{
    // A few example members
    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }

    // Virtual method
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Code to draw a circle...
        Console.WriteLine("Drawing a circle");
        base.Draw();
    }
}
public class Rectangle : Shape
{
    public override void Draw()
    {
        // Code to draw a rectangle...
        Console.WriteLine("Drawing a rectangle");
        base.Draw();
    }
}
public class Triangle : Shape
{
    public override void Draw()
    {
        // Code to draw a triangle...
        Console.WriteLine("Drawing a triangle");
        base.Draw();
    }
}
```

Figura 7 – Polimorfismo | Fonte: MICROSOFT, 2022.

Antes de prosseguir quero dar mais um exemplo de polimorfismo. Iremos criar uma classe genérica de monstros e duas derivadas desse: 1) Saci; 2) Caipora. Colocaremos a classe animal como filha do "MonoBehaviour", e o de Saci e Caipora como filhas de monstros. Usando o polimorfismo, sobrescrevemos o habitat de cada monstro (Figura 8).

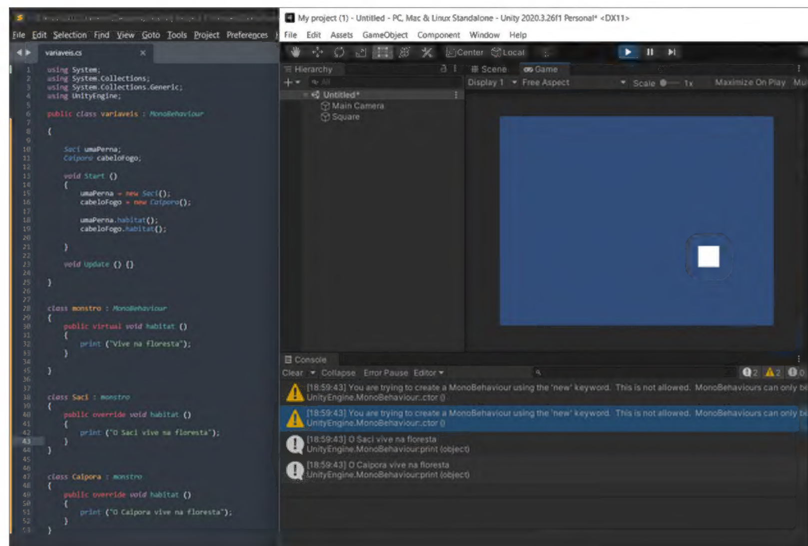


Figura 8 – Polimorfismo Monstros | Fonte: Autor, 2022.

Espero que tenham gostado e até mais! Vida longa e próspera!

Atividade Extra

Aprofunde-se assistindo ao TEDx de Zach Latta e aprenda mais sobre "The Art of Code", ou seja, a arte do código. Zach Latta é cofundador e diretor executivo do Hack Club, onde trabalha para trazer clubes de programação para escolas de ensino médio em todo o mundo. Apesar de estar em inglês você pode ativar a legenda e traduzi-la.

Disponível no Youtube.

Referência Bibliográfica

HENRIQUE, J. **POO: o que é programação orientada a objetos?** São Paulo: Alura, 2019. Disponível em: <https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=Cj0KCQiA_8OPBhDtARIsAKQu0gboagE1jwWzJdVp3JGBZ-nSElc-2JKR0RS7W4PYduVqCTfdzZchtHkaAoMiEALw_wcB>.

MICROSOFT. **Guia do C#.** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/>.

MORAES, M. R. **Curso a Distância**: Introdução a Programação Orientada a Objetos em PHP. São Paulo: CPS, 2010. Acesso em: <http://www.cpscetec.com.br/adistancia/poo_php/aula4.html#:~:text=Algumas%20vantagens%20para%20ju

NASCIMENTO, W. **Jogos 2D com Unity e C#.** São Paulo: Udemy, 2022.



Atividade Prática: Prática de Programação C# II

Título da Prática: Prática de Programação C# II

Objetivos: Usando um input de acelerômetro fazer que uma bola se encaixe em um buraco.

Materiais, Métodos e Ferramentas: Para realizar esta prática vamos utilizar o software Unity, um smartphone Android e acesso ao Youtube.

Atividade Prática

Roteiro

1º Passo – Leia o texto do material de apoio.

2º Passo – Assista aos vídeos da disciplina e aula 12.

3º Passo – Assista ao Tutorial para instalação do Unity Remote disponível nos links:

- **Vídeo 1 - Configuração Unity Hub:** https://youtu.be/svC9uEd1xC0;
- **Vídeo 2 - Configuração MOBILE:** https://youtu.be/OuJlc14vz_U.

4º Passo – Instale o Unity Remote e prepare o seu Unity para os Inputs do Smartphone.

5º Passo – Baixe as sprites disponível em https://bit.ly/3t3aimq.

6º Passo – Importe as sprites para o Unity, arrastando de uma janela com a pasta onde contém os arquivos e arrastando para dentro do Project na pasta de Sprites.

7º Passo – Crie um arquivo de código C# e digite o código para que funcione o acelerômetro (Figura 1)



Figura 1 – Acelerômetro

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

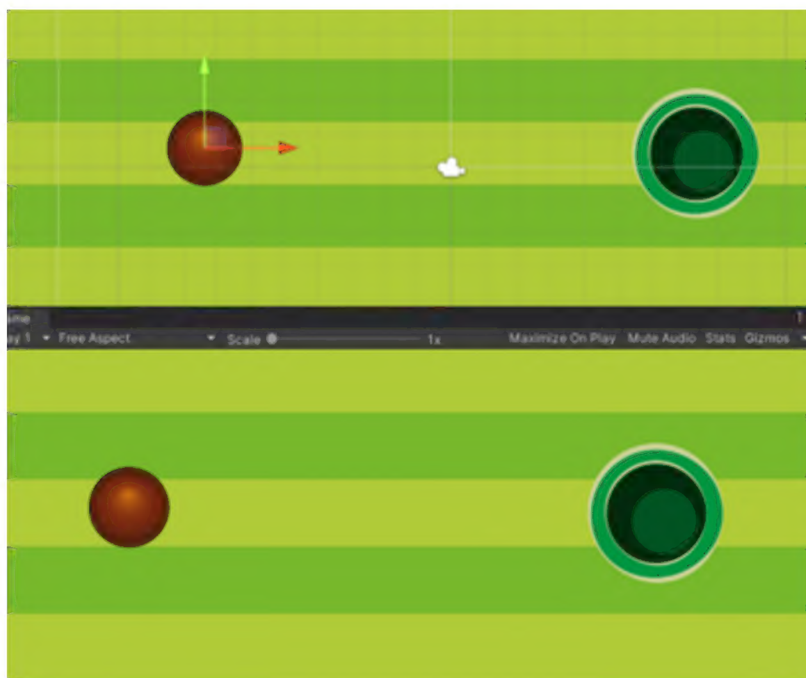
Script do Unity (1 referência de ativo) | 0 referências
public class accelerometer : MonoBehaviour
{

    Mensagem do Unity | 0 referências
    void Update()
    {
        transform.Translate(Input.acceleration.x, Input.acceleration.y, 0);
    }
}
```

Fonte: Autor, 2022.

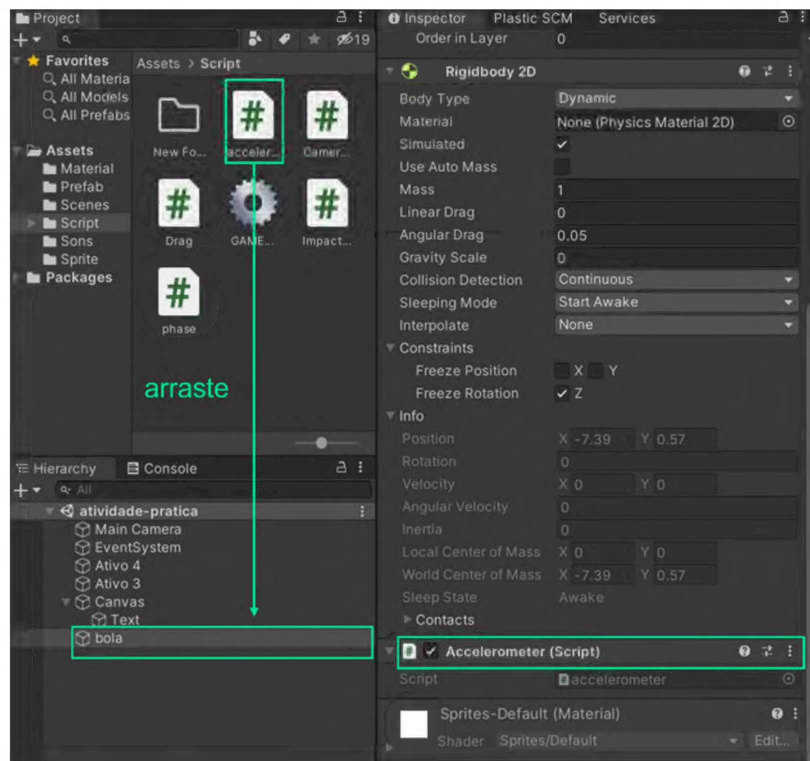
8º Passo – Monte a cena (Figura2) com as sprites colocando na bola o script do acelerômetro (Figura 3).

Figura 2 – Montagem da Cena



Fonte: Autor, 2022.

Figura 3 – Cinculando Script ao Objeto



Fonte: Autor, 2022.

9º Passo – Responda as perguntas.

Após a conclusão da Atividade poste o resultado e responda:

- Conseguiu fazer funcionar o acelerômetro? Como você percebeu a experiência?
- O que você melhoraria no código?

Gabarito Atividade Prática

1) Se você fez o passo-a-passo deu tudo certo. Você vai conseguir controlar a bola com o movimento do celular, mas não conseguirá encaixar a bola no buraco, para que isso ocorra será necessário acrescentar outros códigos.

2) Eu particularmente acrescentaria os códigos para fazer o buraco funcionar.

Ir para exercício